SQL Server Community
Tech Note

sponsored by embarcadero

Maximum Performance and Productivity Series

Five Indexing Tips to Improve SQL Server Performance

By Pinal Dave, Creator of SQL Authority (blog.sqlauthority.com)

January 2012

About Pinal Dave

Pinal Dave is a Microsoft Technology Evangelist (Database and BI). He has written over 2000 articles on the subject on his blog at http://blog.sqlauthority.com. Along with 8+ years of hands-on experience he holds a Master's of Science degree and a number of certifications, including MCTS, MCDBA and MCAD (.NET). He is the co-author of three SQL Server books - SQL Server Programming, SQL Wait Stats and SQL Server Interview Questions and Answers. Prior to joining Microsoft he was awarded the Microsoft MVP award for three continuous years for his contributions to the community.

Introduction

SQL Server Performance is always one of the most challenging subject. Hard drives are getting cheaper and cheaper and data is growing exponentially. With this new pattern we all have an important challenge. In the database world, though, we now have a new problem of performance. Here are few questions that keep coming up in various systems in the industry:

1. Query which was running quickly now taking too long
2. Our report is now running very slow
3. During data import everything gets slow
4. Every day at during a certain time period we are facing many deadlocks
5. Queries are continuously time-outs
6. SELECT queries are slower when INSERT, UPDATE and DELETE are happening
7. … and many more…

Throughout the years I have seen and solved many similar issues. I have seen many bad practices implemented in SQL Server at the server and database level. There are hundreds of tips that one can practice to keep a database at optimal performance.

The purpose of this document is to highlight a few best practices that can give maximum benefits to the SQL Server system. Out of thousands of the best practices I have selected the five best practices related to Indexes.

Drop Unused Indexes

Indexes are commonly created to gain additional performance from the system. It is very common for a new DBA to inherit index systems. When a new DBA gets a system which has been there for long time, there are always lots of indexes already created. New DBAs often do not have the understanding or documentation about why all of those indexes were created. The new DBA cannot drop indexes created by earlier admins and create a few more to accommodate new requests. This leads to the issue that there are way more indexes than needed.

Lots of unused indexes are an extra burden on SQL Server. Every time any field is updated which is referenced in the index, the index also has to be updated. Updating the index is an additional load on the SQL Server engine. If you have an Index maintenance script, it will also be wasting some resources on rebuilding/reorganizing indexes.

The best practice for unused indexes is to drop them. When dropping indexes, one has to be very careful that they do not drop any index which is useful to queries by mistake. It is recommended to evaluate this unused index script on the server after running it continuously for a while without restarting the services or server.

Unused Index scripts are based on DMV and will return lots of results. Select the top 2-3 indexes at a time and drop them on the development server. Keep your server on watch for a while and, if you find it appropriate, drop them on the production server too.

You can download the script from here: http://bit.ly/UnusedIndex

CREATE MISSING INDEXES

Indexes are created to improve performance. As important as it is to drop unused indexes, it is also important to create missing indexes. It is quite normal for developers to keep on writing optimal queries and changing the queries to suit new requests of the end users. The requirements of the end users always keep on changing and developers follow up by changing the query.

The ever changing queries have a circular effect on the indexes. The indexes which are created for specific queries become lout-dated when the underlying queries change. As discussed in the earlier topic, we should drop the unused index. However, we must not forget to check the most relevant indexes against the recently running queries.

Missing index scripts provide the details of the indexes which are most beneficial to the queries. When any query plan is generated it is always looking for the most optimal index. When the most optimal index is not found, SQL Server Engine uses another index which is the next best choice. It is good idea to create an index which is going to be the most effective one.

Missing Index script is based on DMV and will return lots of results. Select the top 2-3 indexes at a time and create it on development server. Keep your server on watch for a while and, if you find it appropriate, create them on production server too.

You can download the script from here: http://bit.ly/MissingIndex

REMOVING DUPLICATE INDEXES

Just like unused and missing indexes – duplicate indexes are another very important concept one must consider. As most developers inherit the database from previous developers, the understanding of the database schema and indexes is not usually at its best. Quite commonly, when the new index is needed developers do not look at the definition of the

existing indexes; they just create the new index as per their requirement. This can lead to multiple indexes with the same definition in the system.

There is absolutely no point to have two indexes with exactly the same structure in any database system. This duplicate index not only takes up space on the hard drive but also reduces the performance. All the INSERT, UPDATE and DELETE queries will have to now update two similar sets of the data on every single occurrence. As there are duplicate indexes, only one of the indexes is used when any query is executing, making the duplicate index redundant.

The best practice is to drop the duplicate index and keep any database free from additional overhead. Again, please be sure to verify that the index is indeed a duplicate before dropping it.

You can download the script from here: http://bit.ly/DuplicateIndex

SIGNIFICANCE OF CLUSTERED INDEXES

More than best practices it is very important to understand the significance of the clustered index. The common opinion and generic best practice suggests that any table should have a clustered index. If your table is very small and the database is of insignificant size, this property can be ignored, however, I suggest understanding the best practice related to clustered indexes.

Another common practice is to create clustered indexes on the columns which are often searched in the database. An additional advantage is that when the first set of the data is retrieved the next set of the data which is commonly queried are placed adjacent. This can help improve query performance. Clustered indexes also help improve performance when the unique values are retrieved.

Here are a few best practices for clustered indexes:

- Columns which have large numbers of unique, distinct data may be good candidates for clustered indexes.
- In OLTP workload the common practice is to create clustered indexes on the primary key as data is often looked up using the same keys. (In SQL Server when a Primary Key is created it automatically creates the clustered index if it does not already exist. There are some rare cases when Primary Keys are on different columns than Clustered indexes). It is indeed good practice to have clustered indexes on unique values (e.g. Primary Keys) as it will avoid adding an additional unique identifier on the clustered index.
- Keep the width of the Clustered Index as narrow as possible.

ADDITIONAL INDEXING BEST PRACTICES

As mentioned earlier there are many best practices related to Indexes. I am listing a few here, in no particular order:

- Primary key is usually a good candidate for clustered index.
- Keep the width of the index as narrow as possible.
- GUID is not a candidate for clustered index (except in extreme cases) as it may lead to higher fragmentation and reduced performance.
- Any column which has high distinctive values (i.e. column is increasing, unique, identity column) is a good candidate for clustered index.
- When creating multiple column indexes it is usually a good idea to have the most selective columns the first column in your index. (Verify this by a thorough test; this can vary case by case. When there are no other external conditional influences this is considered as best practice.)
- Do not add indexes on every single column in the table, indexes should be carefully analyzed before creating.
- Fill Factor 0 or 100 is the default value for the server. The index can be adjusted to the appropriate value. Higher fill factors for less frequently changed data and lower fill factors for more frequently changed data is recommended.

FINAL NOTE

Try all the queries on the development server first. Test all the changes first on the development server and validate all the results. Deploy only after careful consideration. Take all the advice here as a best practice but not as a strict rule.

Embarcadero Technologies, Inc. is the leading provider of software tools that empower application developers and data management professionals to design, build, and run applications and databases more efficiently in heterogeneous IT environments. Over 90 of the Fortune 100 and an active community of more than three million users worldwide rely on Embarcadero's award-winning products to optimize costs, streamline compliance, and accelerate development and innovation. Founded in 1993, Embarcadero is headquartered in San Francisco with offices located around the world. Embarcadero is online at www.embarcadero.com.